# CONCURRENT DATA ENTRY FOR A PORTABLE DEVICE

## PRIORITY

The present application claims priority to U.S. Provisional Patent Application No. 60/516,385, filed on October 31, 2003, titled "Concurrent Data Entry for a Portable Device," the entire disclosure of which is incorporated by reference herein.

## FIELD

The present invention relates to data entry on a portable device, and more particularly, to a system and method of concurrent data entry for a portable device, such as a mobile phone.

## BACKGROUND OF THE INVENTION

Most mobile phones are equipped with a simple 12-button keypad, similar to the keypad 100 shown in FIG. 1. Such a keypad is an inherently poor tool for generating phrases for a 26-letter alphabet. Using traditional text-entry techniques, such as *MultiTap*, an average text message of 7 words requires roughly 70 key presses. The GSM Association (www.gsmworld.com) estimates that in 2003, nearly 500 billion text messages will be sent worldwide from mobile phones. Using current text-entry techniques, this would require approximately 35 trillion key presses. While research has gone into devising a variety of more efficient text input techniques, none has yet emerged as a new standard.

Entering text from the 26 character English alphabet (or practically any other Roman alphabet) using the standard 12-key (0-9,*,#) mobile phone keypad forces a mapping of more than one character per key. The typical mapping has keys 2-9 representing either three or four alphabetic characters in addition to the numerals. All text input techniques that use this standard keypad have to somehow resolve the ambiguity that arises from this multiplexed mapping. The problem may be characterized as involving two main tasks necessary for entering a character: between-group selection of the appropriate group of characters, and within-group selection of the appropriate character within the previously chosen group. Most text input techniques to date can generally be divided into two categories: those that require multiple presses of a single key to make between-group followed by within-group selections, and those that require a single press of multiple keys to make these selections. Because both categories require consecutive key presses, the research focus has been on reducing the average number of key strokes per character ("KSPC") required to enter text. Advances in the area generally make language specific assumptions to "guess" the desired within-group character, thus reducing or eliminating the key presses required for the within-group selection. The success of these techniques, however, is based almost entirely on how closely the text entered conforms to the underlying language model. Given that text entered on mobile phones often involves significant abbreviations and even evolving new "languages" by frequent users of SMS messaging, making language assumptions may not be the best approach to solving the text input problem.

A small number of mobile phones today utilize QWERTY style keypads that enable text entry with techniques similar to typing on a regular keyboard, albeit at a much smaller physical scale (e.g., the Nokia 5510, www.nokia.com). More recently, hybrid devices that combine phones with Personal Digital Assistants ("PDAs"), such as the Handspring Treo (www.handspring.com) and PocketPC Phone (www.microsoft.com), utilize pen-based text input techniques common to PDA's such as Palm's Graffiti (www.palm.com). While these devices are making small inroads into the mobile phone market, the vast majority of mobile phones are equipped with the standard keypad, which has 12 keys: 0-9, *, and #.

Entering text from a 26 character alphabet using this keypad forces a mapping of more than one character per button of the keypad. A typical mapping has keys 2-9 representing either three or four characters, with space and punctuation mapped to the other buttons. All text input techniques that use this standard keypad have to somehow resolve the ambiguity that arises from this multiplexed mapping. There are three main techniques for overcoming this ambiguity: *MultiTap*, two-key, and linguistic disambiguation.

## 1. *MultiTap*

*MultiTap* works by requiring the user to make multiple presses of each key to indicate which letter on that key is desired. For example, the letters pqrs traditionally appear on the "7" key. Pressing that key once yields "p", twice "q", etc. A problem arises when the user attempts to enter two consecutive letters on the same button. For example, tapping the "2" key three times could result in either "c" or "ab". To overcome this, *MultiTap* employs a time-out on the button presses, typically 1-2 seconds, so that not pressing a button for the length of the timeout indicates that you are done entering that letter. Entering "ab" under this scheme has the user press the "2" key once for "a", wait for the timeout, then press "2" twice more to enter "b". To overcome the time overhead this incurs, many implementations add a "timeout kill" button that allows the user to skip the timeout. If we assume that "0" is the timeout kill button, this makes the sequence of button presses to enter "ab": "2-0-2-2". *MultiTap* eliminates any ambiguity, but can be quite slow, with a keystrokes per character (KSPC) rate of approximately 2.03. (See MacKenzie, I.S. (2002). KSPC (keystrokes per character) as a characteristic of text entry techniques. *Fourth*

*International Symposium on Human-Computer Interaction with Mobile Devices*. p. 195-210.)

## 2. Two-Key Disambiguation

The two-key technique requires the user to press two keys in quick succession to enter a character. The first keypress selects the appropriate group of characters, while the second identifies the position of the desired character within that group. For example, to enter the character "e", the user presses the "3" key to select the group "def", followed by the "2" key since "e" is in the second position within the group. This technique, while quite simple, has failed to gain popularity for Roman alphabets. It has an obvious KSPC rate of 2.

## 3. Linguistic Disambiguation

There are a number of linguistic disambiguation schemes that utilize knowledge of the language to aid the text entry process. One example is *T9* (see www.tegic.com), which renders all possible permutations of a sequence of button presses and looks them up in a dictionary. For example, the key sequence "5-3-8" could indicate any of 27 possible renderings (3x3x3 letters on each of those keys). Most of these renderings have no meaning, and so are rejected. Looking each of them up in a dictionary tells the system that only "jet" is an English word, and so it is the one rendered. Ambiguity can, however, arise if there is more than one valid rendering in the language, in which case the most common is presented. For example, the sequence "6-6" could indicate either "on" or "no". If the system renders the wrong word, a "next" key allows the user to cycle through the other valid permutations. An analysis of this technique for entering text from an English corpus found a KSPC close to 1 (see MacKenzie, I.S. (2002) "KSPC (keystrokes per character) as a characteristic of text entry techniques," *Fourth International Symposium on Human-Computer Interaction with Mobile Devices*, pp. 195-210). Newer linguistic disambiguation techniques such as *LetterWise* and *WordWise* (see www.eatoni.com) also perform similarly well, with subtle advantages over earlier techniques. While these all have excellent KSPC rates, the success of linguistic-based systems depends on the assumption that users tend to enter "English-like" words when sending text messages. However, users often use abbreviations and incomplete English when text messaging. Further, users of text messaging often communicate in acronyms or combinations of letters and numbers (e.g., "b4" for "before"). Another problem with these linguistic techniques is that users have to visually monitor the screen in order to resolve potential ambiguities, whereas the *MultiTap* and two-key techniques can be operated "eyes-free" by skilled users.

***Using Tilt Sensors in Portable Devices***

Attempts have been made to incorporate tilt sensors in portable devices. *Unigesture* uses tilt as an alternative to button pressing, eliminating the need for buttons for text entry. (See Sazawal, V., Want, R., & Borriello, G. (2002), "The Unigesture Approach. One-Handed Text Entry for Small Devices," *Mobile HCI*, p. 256-270.) Rather than having the user make one of 8 ambiguous button presses (as is the present case with mobile phones), *Unigesture* has the user tilt the device in one of 7 directions to specify the group, or "zone", of the character that is desired. The ambiguity of the tilt is then resolved by using dictionary-based disambiguation.

*TiltType* combines button pressing and tilt for entering unambiguous text into a small, watch-like device with 4 buttons. (See Partridge, K., Chatterjee, S., Sazawal, V., Borriello, G., & Want, R. (2002), "TiltType: accelerometer-supported text entry for very small devices," *ACM UIST Symposium on User Interface Software and*

*Technology*, pp. 201-204.) Pressing a button triggers an on-screen display of the characters that can be entered by tilting the device in one of eight directions. The user then makes the appropriate tilt and releases the button.

***Chording Keyboards***

Chording keyboards typically have a series of chording keys that can be used in conjunction with a keypad to enter text. Two-handed chorded keyboards have been used by the U.S. postal service for mail sorting, and are still used today by stenographers. The Twiddler (see www.handykey.com) and the Septambic Keyer (see wearcam.org/septambic/) are examples of modern-day one-handed chording keyboards. Designed to be held in the hand while text is being entered, both are commonly used as part of a wearable computer. The Twiddler is equipped with 6 keys to be used with the thumb, and 12 for the fingers, while the traditional Septambic Keyer has just 3 thumb and 4 finger switches. The Septambic Keyer allows for 47 different combinations of key presses, while the Twiddler allows over 80,000, though not all keys are used for text entry.

None of the approaches described above have been commercially successful. What is needed is an efficient system and method for entering data into a numeric keypad of a portable device.

## DETAILED DESCRIPTION

In view of the wide variety of embodiments to which the principles of the present invention can be applied, it should be understood that the illustrated embodiments are exemplary only, and should not be taken as limiting the scope of the present invention.

FIG. 2 is a pictorial representation of a mobile phone 200 according to an exemplary embodiment of the present invention. The phone 200 includes a 12-button keypad 202, a display 204, and an antenna 206. The phone 200 is also likely to include a microphone and speaker, which are not shown in FIG. 2.

A user may tilt the phone 200 along a first axis 206 and/or a second axis 208 to assist in entering text data into the phone 200. By determining the tilt of the phone 200 as the user is pressing a button on the keypad 202, the phone 200 is able to combine a first concurrent input (button press) with a second concurrent input (tilt status) to identify an entered character. Thus, the number of unique characters that may be entered is greater than the number of buttons (and the number of detectable tilt states).

In a preferred embodiment, the first and second axes 208 and 210 are chosen so that as a user holds the phone 200 with the display 204 facing the user, the first axis 208 runs through the left and right sides of the phone 200, so that rotation about the first axis 208 produces a forward and backward tilt. The second axis 210 preferably runs through the top and bottom of the phone 200, so that rotation about the second axis 210 produces a side-to-side tilt. Selecting the axes 208 and 210 in this way, as opposed to selecting an axis coming perpendicular out of the face of the phone 200, will allow the user to generally view correctly oriented text on the display 204 (i.e., the text will generally run horizontally from the user's frame of reference).

Rotation about an axis coming perpendicular out of the face of the phone 200 would require the user to read text at an angle. Comfort and ease of operation may also help in determining the axes of rotation.

In most embodiments, the amount of tilt required for disambiguation is selected so that the user will still be able to easily view the keypad and display at all tilted states. Therefore, while a 90 degree tilt to the left may be detected as a disambiguating tilt, a more reasonable minimum tilt might be 5-10 degrees, with an average expected tilt of about 30 degrees. The amount of tilt should be large enough to avoid erroneous disambiguation, but small enough to promote comfort, speed, and ease in viewing the display and keypad.

FIG. 3 is a simplified block diagram illustrating a system 300 for concurrent data entry in a portable device, such as a mobile phone. The system 300 includes a microprocessor 302 having connections to a tilt sensor 304 and a keypad 306. The system also includes a display 308 to allow a user to view entered text and other information, such as a message received from another mobile user.

The microprocessor 302 operates using software 310 and memory 312. The software 310 preferably runs on an operating system associated with the system 300, such as a mobile phone operating system. The software may include one or more modules such as a tilt/text engine 314 operable to identify entered data by using tilt and button presses as inputs. The software may be written in Java, C++, or any other suitable language.

The memory 312 may include a sample record stack 316 to store a recent samples obtained by the microprocessor 302 from the tilt sensor 304. This may be useful in determining tilt relative to previous orientations, such as if a relative tilt implementation is used. A relative tilt embodiment is described in greater detail below. The number of samples to be stored in the sample record stack 316 will likely depend on a number of factors, including the available memory, the sampling rate, and possibly a user defined setting that may be changed as the user becomes more proficient (and quicker) entering data using button presses concurrently with tilt.

In an alternative embodiment, the tilt sensor 304 could make use of a digital camera, such as one contained in a "camera phone" to help in determining tilt. For example, by identifying a visual pattern in a first image and identifying how the visual pattern has changed in a second or other subsequent image, the change in orientation (or tilt) of the system 300 may be determined. This alternative technique may be useful in an environment subject to underlying background accelerations that might be perceived as accelerations (or tilts) by the tilt sensor 304. For example, a user entering data on an accelerating bus may observe effects due to the tilt sensor sensing acceleration from the bus in addition to acceleration from an intentional tilt.

FIGs. 14 and 15 illustrate a technique for using computer vision to determine tilt. Using a camera built-in to a mobile phone, tilting can be detected without the need of an accelerometer. By monitoring shifts in the camera's visual field, tilting and panning movements can be inferred: shifts in the image indicate a shift in the camera's position, but in the opposite direction. In FIG. 15, the image within the visual field of the camera has shifted to the right, when compared to FIG. 14. This indicates a left lateral move of the camera, from which subtle tilting gestures can be inferred.

While the embodiment of FIG. 3 makes use of software for a majority of operations, a hardware or firmware implementation could also be used. For example, a series of AND/OR gates implemented with transistors and/or programmable logic circuitry.

FIG. 4 is a series of pictorial diagrams illustrating an exemplary embodiment for determining tilt in a system for concurrent data entry in a mobile phone. The series of figures show one possible combination of tilt axes that may be used to disambiguate the meaning of button presses. Tilting the phone to the left (left diagram) selects the first letter of the button, tilting the phone away from the body (top diagram) selects the second letter of the button, tilting to the right (right diagram) selects the third letter, and, if a fourth letter is present on the button, tilting towards the user's body (bottom diagram) selects the fourth letter. Pressing a key without tilting (center diagram) results in entering the numeric value of the key. Space and backspace operations may be carried out by pressing unambiguous single-function buttons.

Supporting both lowercase and uppercase characters would require a further disambiguation step since a total of seven characters per key would need to be mapped for keys 2-6 and 8, and nine characters each for the 7 and 9 keys. Adding case sensitivity could be done by either requiring the pressing of a "sticky" shift-key, or considering the magnitude of the tilt as a disambiguator where greater magnitude tilts result in upper case letters. FIG. 5 illustrates the latter of these two techniques for the "7" key and the letters "P", "Q", "R", and "S" on a mobile phone. Eyes-free entry, however, would likely be more difficult using the technique shown in FIG. 5, since the user may wish to confirm that a large enough tilt was used to realize a capital letter.

The system uses the standard 12-button mobile phone keypad augmented with a low-cost tilt sensor. The system uses a combination of a button press and tilting of the device to determine the desired letter. This technique differs from *TiltType* (Partridge et al. (2002), "TiltType: accelerometer-supported text entry for very small devices," *ACM UIST Symposium on User Interface Software and Technology*, pp. 201-204.) in the size of the device, the type of keypad used, ease of one-handed operation, and in the sensing algorithms, described in detail below. The standard phone keypad mapping assigns three or four alphabetic characters and one number to each key. For example, the "2" key also has the characters "a", "b", and "c" assigned to it. The system assigns an additional mapping by specifying a tilt direction for each of the characters on a key, removing any ambiguity from the button press. The user presses a key while simultaneously tilting the phone in one of four directions (left, forward, right, back) to input the desired character. For example, pressing the "2" key and tilting to the left inputs the character "a", while tilting to the right inputs the character "c". By requiring only a single keypress and slight tilt to input alphanumeric characters, the overall speed of text entry can be increased. Further, unlike some techniques that improve on the status quo *MultiTap* technique, the system is not language dependent, and thus can be used without visually attending to the display screen.

## *Techniques for Calculating Tilt*

The tilt of the phone is taken as whichever direction has the greatest tilt relative to an initial "origin" value. Described herein are three alternative embodiments for determining the tilt value: *key tilt*, *absolute tilt*, and *relative tilt*.

### a. Key Tilt

In a first embodiment, the amount of tilt is calculated as the difference in the value of the tilt sensors at key down and key up. This requires the user to carry out three distinct movements once the button has been located: push the button, tilt the phone, and release the button. A similar approach has been used with a watch-like four-button device. (See Partridge, K., Chatterjee, S., Sazawal, V., Borriello, G., & Want, R. (2002), "TiltType: accelerometer-supported text entry for very small devices," *ACM UIST Symposium on User Interface Software and Technology*, pp. 201-204.) Initial experiments using a key tilt implementation on a 12-button mobile phone keypad showed that this implementation was much slower than the traditional *MultiTap* technique.

### b. Absolute Tilt

In a second embodiment, the tilt sensor's value at any given time is compared to a "fixed" absolute origin. Only two distinct movements are required to enter a character: the phone is tilted and then a key is pressed. In contrast, the key tilt embodiment requires three movements: a key is pressed, the phone is tilted, and then the key is released.

However, users do not typically maintain a constant arm posture. Thus, in order for the tilt value to be meaningful, the fixed origin will preferably be reset every time the user's gross arm posture changes.

Further, when using the system to enter two characters requiring tilt in opposite directions, more movement is required using this absolute approach, since the first tilt must be undone, then the new tilt applied. For example, entering the letters "ac" using the "2" key requires an initial tilt of some angle $\alpha$ to the left to enter the "a". Then, the user has to tilt the same angle $\beta$ in the reverse direction to return to the origin, before tilting another angle $\beta$ to the right to enter the letter "c". The total amount of movement is $2\beta + \alpha$, instead of the smaller $\alpha + \beta$ that one may expect. However, one advantage of this embodiment over the key tilt embodiment is that if successive characters with the same tilt direction are to be entered, then the user can keep the phone tilted at that direction for the successive keypresses.

### c. Relative Tilt

According to a third embodiment, tilt is calculated relative to a floating origin that is set when a tilt gesture begins. The beginning of a gesture is determined by continuously watching for a change in orientation or a change in the direction of a tilting gesture. This approach solves both problems of the absolute tilt embodiment. Since all tilts are relative to the beginning of the gesture, there is no absolute origin that need be reset when changing arm position. Further, opposite-direction tilts do not require double tilting, since the second tilt's origin is the end of the first tilt's gesture. So, entering the letters "ac" requires a tilt of some angle $\alpha$ to the left to enter a, then another tilt of angle $\beta$ to the right to enter the c, for a total movement of $\alpha + \beta$. Note that, like with the absolute tilt embodiment, when entering only letters, we can enter successive characters with the same tilt direction without re-tilting the phone, by looking at the last significant tilt.

## Disambiguating Characters

Once a tilt state has been determined for a pressed button, a character associated with the pressed button may be identified by referring to a tilt menu that specifies tilt states that correspond to particular characters. The tilt menu may be, for example, a simple lookup table stored in memory. Alternatively, disambiguation may be performed in hardware, such as in dedicated logic circuitry.

## Experimental Verification

To confirm that embodiments of the present invention do indeed provide speed and efficiency benefits over the current *MultiTap* technique, an experiment was conducted, as described below.

## Hardware

A Motorola i95cl mobile phone equipped with an Analog Devices ADXL202EB-232 2-axis accelerometer board to enable tilt sensing served as the test device. The accelerometer board was connected to the phone via a serial cable (with the addition of an external power line). While a preferable commercial implementation is likely to have the tilt-sensing circuitry enclosed within the casing of the mobile phone, the external serial-cable mounting served to provide an indication of expected results.

An implementation of a relative tilt system would require regular sampling from the tilt sensor. Because the experimental hardware provided for a reliable sampling rate of only approximately 10 Hz, an absolute tilt approach to tilt determination was used. To implement a relative tilt technique, at least a 20-50Hz sampling rate should be used.

Under the absolute tilt experimental setup, the user was allowed to reset the origin at any time by holding the phone at the desired orientation and pressing "0". The additional movement required by this approach, however, was believed to be acceptable for evaluation purposes because if the experimental system performed well despite this additional movement, then any more robust implementation using a relative tilt approach would likely only perform better. In other words, the evaluation was biased against the experimental system.

Because the ADXL board was able to detect a tilt of only fractions of a degree, only a very small tilt of the phone was necessary to disambiguate a button press. The maximum of the tilt in either axis was taken to be the intended tilt, with a 10% bias towards forward/back. This bias was included due to a tendency of users to pitch to the dominant side when tilting forward with the wrist.

## Software

The software to read tilts and render text, as well as conduct the experiment, was written in Java 2 Micro-Edition using classes from both the Mobile Devices Information Profile (MIDP 1.0) and proprietary i95cl specific classes.

The experiment was conducted entirely on the mobile phone rather than simulating a mobile phone keypad on some other device. All software, including those portions implementing the text entry techniques and data presentation and collection ran on the phone. No connection to an external computing device beyond the tilt sensor was used.

The *MultiTap* implementation set up for comparison used the i95cl's built-in *MultiTap* engine, with a 2 second timeout and timeout kill. We only considered lowercase text entry in this evaluation. As such, the *MultiTap* engine was modified slightly to remove characters from the key mapping that were not on the face of the button, so that the options available were only the lower case letters and numeral on the key.

## Procedure

Experiment participants (5 men and 5 women of whom 3 were left-handed and 7 right-handed, none of which had any experience composing text using either technique) entered short phrases of text selected from among those in MacKenzie's

English phrase dictionary (see www.yorku.ca/mack/phrases2.txt). The desired text phrases were shown to participants on the screen on the phone.

Timing began when participants entered the first character of the phrase, and ended when the phrase was entered completely and correctly. If an erroneous character was entered, the phone alerted the user by vibrating, and the user was required to correct their error. With this procedure, the end result was error-free in the sense that the correct phrase was captured. Also, the phrase completion time incorporates the time taken to correct for errors.

Before beginning each treatment, participants were told to read and understand the displayed phrase before entering it, and were given instructions for that treatment as follows:

*MultiTap* instructions: to enter a character using the *MultiTap* technique, first find the key that is labeled with that character. Press that key repeatedly until the desired character is reached. Press once for the first character, twice for the second, three times for the third, and, if present, four times for the fourth. Once you have found the correct letter, and are ready for the next one, you simply repeat the process. If the letter you wish to enter next is on the same key, you must first either press the "right" arrow on the phone or wait two seconds for the cursor to advance.

Experimental system instructions: the technique works by tilting the phone in the direction of the letter you wish to enter, then pressing the key on which it is inscribed. For the first letter, tilt left. For the second letter, tilt forward. For the third letter, tilt to the right. For the fourth letter, tilt towards you. The direction of tilt is measured relative to the "centre" or "origin" position of the phone. You can reset the origin at any time by pressing the "0" key.

The experimenter then demonstrated the relevant technique. To ensure that participants understood how the technique worked, they were asked to enter a single phrase that would require tilting in all four directions for the experimental system, or two successive letters on the same key for *MultiTap*.

Additional instructions were given for both techniques to describe space and delete keys, as well as to enter an extra space at the end of the phrase to indicate completion. The process for error correction was also explained to them. Participants were also directed to rest as they liked between phrases, but to continue as quickly as possible once they had started entering a phrase.

**Results**

The data collected from 10 participants took an average of 10.3 minutes per block. A total of 145360 correct characters of input were entered for the 6400 phrases.

**Text Entry Speed**

We use the standard wpm (words-per-minute) measure to describe text entry speed. This is traditionally calculated as characters per second * 60 / 5. Because timing in our experiment started only after entering the first character, that character was not included in calculations of entry speed. Thus, for the purposes of these computations, the length of a phrase is n-1 characters. Also, to signify completion, users had to enter an extra space at the end of each phrase. However, entry of the last real character of the phrase was considered to be the end time.

The average text entry speed for all blocks were 11.76 wpm and 10.11 wpm for the experimental system and *MultiTap* respectively. Overall, the system was 16.3% faster than *MultiTap*.

The means for the first block of trials were 7.42 wpm and 7.53 wpm, for the system and *MultiTap* respectively. Performance in both techniques increased steadily, with the means for the last (16th) block of trials of 13.57 wpm for the system and 11.04 wpm for *MultiTap*. While subjects performed marginally better with *MultiTap* initially, they improved considerably faster with the experimental system, with the spread between the techniques reaching 22.9% in favor of the experimental system by the end of the experiment.

Analysis of variance indicated significant main effects for technique ($F_{1,8} = 615.8, p < .0001$), and block ($F_{15,120} = 145.2, p < .0001$). There was also a significant technique x block interaction ($F_{15,120} = 20.5, p < .0001$), indicating that participants improved at different rates for the different techniques. FIG. 6 illustrates these effects.

From our analysis and FIG. 7, we see that without prior experience with either technique, the system started out performing worse than *MultiTap*, only crossing over at block 4. This is likely because the system required participants to master two distinctly different motor skills: pressing the key, and tilting the phone. *MultiTap* required only a single type of motor action: multiple presses of the key.

FIG. 8 shows data after participants switched techniques (i.e., the second half of the experiment). We see here that the system starts off faster than *MultiTap*, indicating that participants' were able to take advantage of and transfer their previous experience with *MultiTap* in the first half of the experiment. This is a positive indication since it means that real users with lots of experience with *MultiTap* can transfer at least some of that skill if they switch to the experimental system. Note, however, that there is quite a bit more variability in the performance for the experimental system, as indicated by the poorer fit of the power curve as compared to

FIGs. 6 and 7. This indicates that participants experienced some level of interference due to previous experience with *MultiTap*.

**Error Rates**

Given that the experimental procedure required participants to make corrections as they proceeded, with an end result of a completely correctly entered phrase, the entry speed results discussed previously incorporate the cost of error correction. However, it is still useful to look at a more explicit error rate. We calculate percentage error rate as the number of characters entered that did not match the expected character, divided by the length of the phrase. In this case, we used the actual length of the phrase, and not (n-1) as in the wpm rate.

Overall, error rates were much higher for the experimental system (11%) than for *MultiTap* (3%). This effect was statistically significant ($F_{1,8} = 1378.8, p < .0001$). There was also a significant effect for blocks ($F_{15,120} = 21.1, p < .0001$). A significant technique x block interaction ($F_{15,120} = 23.3, p < .0001$) and FIG. 9 indicate that while the error rates for *MultiTap* remain quite constant throughout the experiment, the error rates for the experimental system drop rapidly over the first 8 blocks, and begin to asymptote from block 9 onwards.

As with the entry time analysis, a significant order x technique interaction ($F_{15,120} = 168.9, p < .0001$) indicates that participants exhibited *asymmetric* transfer effects. An analysis of the first half of the data (i.e., before participants switched techniques) indicates main effects similar to that of the entire dataset: technique ($F_{1,8} = 632.4, p < .0001$), blocks ($F_{15,120} = 7.3, p < .0001$), and technique x block interaction ($F_{15,120} = 10.4, p < .0001$), as shown in FIG. 10. Interestingly, the mean system error rate (8.6%) was lower than for the entire data set, indicating that the lack of interference from *MultiTap* was beneficial.

FIG. 11 illustrates the data from trials in the second half of the experiment (i.e., after participants switched techniques). Comparing this to FIG. 10, the mean the system error rate of 13.5% is much higher than the mean 8.6% rate in the first half of the experiment. Further, the first 8 blocks of trials did not exhibit a constant trend for the experimental system. Clearly, participants' previous experience with the *MultiTap* technique was having a detrimental effect on their ability to use the experimental system right after the switch in technique occurs. This is consistent with the effect observed in the text entry speed data illustrated earlier in FIG. 8. However, this effect wears off roughly after block 8.

To examine the cause of the higher system error rate, errors may be grouped into two categories: *tilt errors* and *button errors*. *Tilt errors* are those where the participant entered a letter that appears on the same button as the correct letter, indicating that an erroneous tilt was made. *Button errors* are those where the participant entered a letter that appeared on a different button.

We had anticipated that *button errors* would have similar rates for the system and *MultiTap*. However, the results showed a significant difference ($F_{1,8}$= 320.67, $p < .0001$), where 3% of characters entered in the *MultiTap* trials were *button errors*, but only 1.5% of characters entered in the experimental system showed this type of error.

Reconciling this low *button error* rate with the high overall error rate for the system, it is clear that most of the errors committed while using the system were *tilt errors*. Breaking down the *tilt error* rate by letter shows that participants committed significantly more errors for some letters than others ($F_{25,200,}$= 2.47, $p < .001$), as Figure 10 illustrates.

Analysis of variance showed a significant main effect for tilt direction on *tilt error* rate ($F_{3,24}$= 37.6, $p < .0001$). Pairwise means comparisons showed a significantly higher *tilt error* rate for those letters requiring forward or backward tilting than those requiring right or left tilting. In particular, backwards tilt results in significantly higher errors than all the other tilt directions. FIG. 12 illustrates this trend.

As was discussed previously, the overall system error rate decreases with practice. As such, it is possible that the high *tilt error* rate for backward tilt (letters "s" and "z") is due to the limited amount of practice users had with entering the letter "z", which was only entered 3 times in the experiment by each participant. However, the other letter that required backward tilting, "s", also showed a similarly high *tilt error* rate, despite being entered very frequently during the experiment. In other words, additional practice did not seem to decrease the backward *tilt error* rate significantly, indicating that users had an inherent difficulty with backward tilting actions. FIG. 13 illustrates tilt error rates as a percentage for each letter in the alphabet.

When participants committed a *tilt error* for a letter requiring a left or right gesture, 82% of the time they ended up entering the forward-tilt letter on that button. This indicates that the 10% bias we introduced in our algorithm seems to overcompensate. Reducing this compensation factor may lower *tilt error* rate for left/right tilts.

The increased error rate for forward/backward movements is possibly explained by limitations of the absolute tilt system used in our experiment. Participants tended to set the origin, then have the phone slowly "creep" forward as they made far more forward than back tilting gestures. As a result, the phone was always tilted somewhat forward. This meant that an exaggerated back gesture was required to enter "s" or "z", which users often failed to accomplish on the first attempt. The tendency to hold the phone in a forward position also explains why most *tilt errors* resulted in entering the forward tilt letter on the same button. Due to hardware constraints, an absolute tilt implementation was used, instead of the more efficient and accurate relative tilt method. Error rates would likely be improved if relative tilt were to be used, since users would not have to tilt a specific amount past the origin between characters as required in the absolute tilt method, and they also would not have to exaggerate the back tilts to overcome the forward posture. These extra requirements are a likely cause of errors, particularly if users attempt to perform the technique quickly without watching the screen. Relative tilt would be more amenable to fast, "eyes-free" use.

The tilt angle required ranges from a little more than 0 degrees to an approximate maximum of 90 degrees. From our observations of participants in our experiment, it appears that the average tilt angle is probably around 30 degrees. With a more definitive determination of this parameter or at least a smaller bound on its range, it would be possible to develop a model that more accurately describes the system than KSPC.

We will also examine whether different tilting directions are more natural for the wrist, and perhaps replace the simplistic "left/right, back/forward" technique examined here. We will also test an implementation that groups letters to minimize the need to re-tilt between character entries.